Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

Grado en Ingeniería Informática

Diseño de un sistema de nuevos periféricos para GameBoy Advance

Autor:
Fabio Banchelli Gracia

Director:
Enric Xavier Martin Rull

Especialidad:
Ingeniería de computadores

Departamento:
ESAII

Abril 2018

Resumen

Este trabajo nace a partir de un proyecto de carácter personal y vocacional. Deriva de otros proyectos anteriores de programación sobre la GameBoy Advance dentro de la asociación VGAFIB. Ninguno de los trabajos previos utilizó el puerto serie de comunicaciones de la consola. El propósito de este Trabajo Final de Grado fue desde el inicio descubrir cómo funciona dicho puerto y qué se puede llegar a conseguir con él. Se utiliza como intermediario entre la consola y otros dispositivos un microcontrolador de la familia PIC.

Aquest treball neix a partir d'un projecte de caràcter personal i vocacional. Deriva d'altres projectes anteriors de programació sobre la GameBoy Advance dintre de l'associació VGAFIB. Cap dels treballs previs va fer servir el port sèrie de communicacions de la consola. El propòsit d'aquest Treball Final de Grau des del principi es descobrir com funciona aquest port i què es pot arribar a aconseguir amb ell. Es fa servir com a intermediari entre la consola i altres dispositius un microcontrolador de la família PIC.

This work is born from a personal and vocational project. It derives from other previous projects in programming for the GameBoy Advance within the VGAFIB association. None of the previous projects used the serial communications port of the console. The main purpose from the start of this Final Project was to uncover how this port behaves and what can it be done with it. A microcontroller from the PIC family is used as a intermediary between the console and other devices.

Índice

Introducción

Motivación

Como miembro de la VideoGames Association FIB (VGAFIB) [1] he tenido amplio contacto con el desarrollo de videojuegos a nivel amateur durante los últimos cuatro años. A lo largo de mi participación como miembro de la asociación encontré especial interés en el desarrollo de videojuegos para consolas, que difiere mucho del desarrollo para un ordenador de sobremesa. Junto con otros compañeros identificamos una consola en concreto con la que habíamos pasado tantas horas de pequeños: la GameBoy Advance (GBA) [2]. Comenzamos por crear pequeños programas que pintaban unos cuantos pixeles en pantalla. Poco a poco fuimos incrementando la complejidad de las applicaciones hasta llegar a algún pequeño juego.

En el contexto de la asociación se intenta promover el desarrollo de videojuegos dentro y fuera de la misma. Una de las dinámicas que utilizamos para darnos visibilidad hacia el exterior son los cursos para alumnos de la FIB. Se realizan una vez al cuatrimestre después de la época de exámenes. Estas actividades son de especial interés tanto para nosotros como profesores como para los alumnos.

Durante el curso 2016/2017 impartimos un curso de introducción a la programación de videojuegos para GBA [3]. Después del éxito de dos ediciones del curso y nuestro creciente interés en la plataforma decidí descubrir quñe mas podía dar de sí la máquina. Entre varias funcionalidades me fijé en la funcionalidad de comunicación de la GBA. Mi idea fue intentar ampliar las funcionalidades de la consola a través de periféricos utilizando el puerto de comunicaciones de la máquina.

Objetivos

Este proyecto que tiene como objetivo diseñar y construir un periférico personalizado para un dispositivo concreto, la GameBoy Advance (GBA). Este dispositivo se comunicará con la consola para implementar funcionalidades de las que la máquina carece (i.e. Detección de Inclinación del sistema, Conexión Internet, Geolocalización...). Posteriormente, se diseñará una pequeña demo para la máquina que utilice la funcionalidades del periférico.

El periférico debe actuar como traductor entre la GBA y otros dispositivos que implementen las funcionalidades añadidas como sensores, microcontroladores y demás. La Figura 1 Muestra un esquema del sistema a diseñar.

Figura 1: Esquema del sistema de periféricos para la GameBoy Advance

Competencias abarcadas

A continuación se listan las competencias marcadas por la FIB y que este proyecto aborda.

- **CEC1**: Diseñar y construir sistemas digitales, incluyendo computadores, sistemas basados en microprocesadores y sistemas de comunicaciones.
  - **CEC1.1**: Diseñar un sistema basado en un microprocesador o un microcontrolador.
  - **CEC1.2**: Diseñar o configurar un circuito integrado usando las herramientas de software adecuadas.
- **CEC2**: Analizar y evaluar arquitecturas de computadores incluyendo plataformas paralelas y distribuidas, y desarrollar y optimizar software para dichas plataformas.
  - **CEC2.2**: Programar considerando la arquitectura hardware, tanto en ensamblador como en alto nivel.
  - **CEC2.3**: Desarrollar y analizar software para sistemas basados en microprocesadores y sus interfícies con usuarios y otros dispositivos.
- **CEC3**: Desarrollar y analizar hardware y software para sistemas empotrados y/o de muy bajo consumo.
  - **CEC3.1**: Analizar, evaluar y seleccionar las plataformas hardware y software más adecuadas para el soporte de aplicaciones empotradas y de tiempo real.

Estado del Arte

La consola contó con una amplia gama de periféricos durante su vida [4] de los cuales destaca el adaptador infrarrojo [5], que permitía la comunicación entre dos sistemas a través de comunicación infrarroja. Este adaptador se conecta a la consola a través del puerto serie. Otros periféricos se incorporan al cartucho que contiene el juego [6]. Estos sistemas son completamente dependientes del juego, por lo que podrían no considerarse periféricos.

La comunidad de seguidores de la GBA ha publicado varios proyectos independientes para la consola [7]. Entre ellos hay tanto software, en forma de videojuegos, como hardware, en forma de sistemas periféricos. Tratándose de una consola que cumple más de 15 años, la documentación es escasa y difícil de encontrar. Las principales fuentes de información son: la guía de programación del fabricante [8], una guía de programación no oficial [9] y la página GBATEK [10], que contiene toda la información recogida por usuarios y desarrolladores. A pesar de no ser una fuente oficial, es la principal referencia de la comunidad.

Destacan los proyectos gpsvario [11] y GameBoy Analog Meter [12]. El primero utiliza un módulo GPS para enviar datos a la consola a través del puerto serie y visualizarlos en pantalla. El segundo conecta la GBA a una placa de desarrollo con un microcontrolador y recibe datos para mostrarlos por pantalla. Ambos proyectos son de carácter lúdico para sus respectivos desarrolladores y no tienen como objetivo incluir una documentación extensa.

Descripción del sistema

Requerimientos

Jugabilidad

Las aplicaciones que se ejecutan en la GBA son videojuegos. Los videojuegos se estructuran en un bucle básico, mostrado en la Figura 2, que debe permitir la interacción del jugador y lo que éste ve en pantalla. Por tanto, la latencia del sistema periférico no puede superar un cierto límite y afectar la experiencia del jugador.

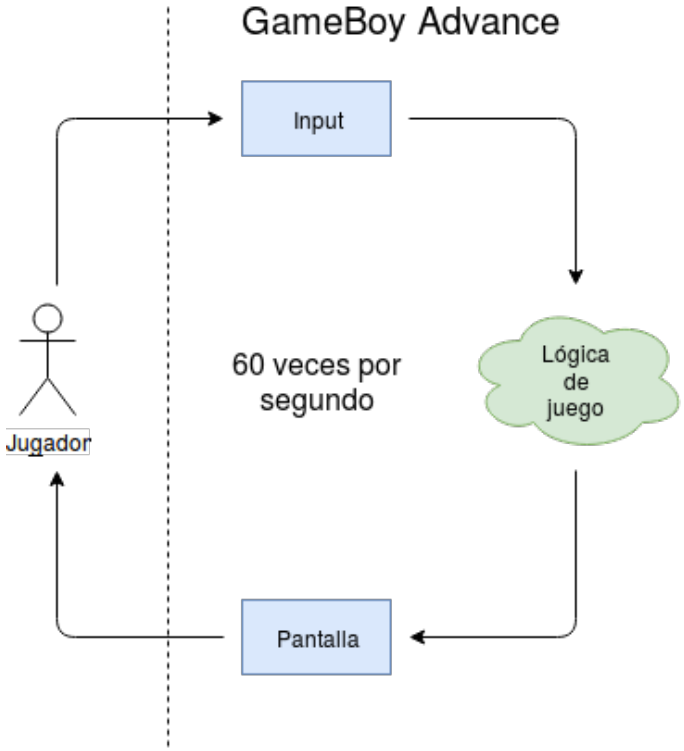

Figura 2: Esquema de la interacción entre el videojuego y el jugador

Portabilidad

Los componentes del módulo periférico deben elegirse teniendo en cuenta que la GBA es un sistema portátil. El factor forma y tamaño debe ser reducido para no comprometer a la portabilidad del conjunto. Es importante considerar también el consumo, pues la batería de la GBA debe alimentar la consola y el periférico.

Selección de un microcontrolador intermediario

Como se discute en secciones posteriores, la GBA soporta un número limitado de protocolos de comunicación. De todos ellos, solo hay uno estándard. Por tanto, será necesario un dispositivo intermediario entre la GBA y los periféricos en caso que los periféricos a utilizar no soportan los protocolos de la GBA. El dispositivo intermediario será un microcontrolador que soporte la comunicación con la GBA y varios protocolos estándard.

La familia de microcontroladores PIC está bastante extendida y cuenta con distintos modelos. Los motivos por los cuales se ha elegido un microcontrolador de esta colección son sencillos: ampliamente utilizados, conocimiento previo por mi parte, disponibilidad inmediata en el laboratorio del departamento de ESAII.

La selección del modelo concreto de PIC pasa por filtrar la lista de productos del fabricante [13]. Los campos principales a filtrar son el número de pines y la frecuencia máxima de la CPU. Por motivos que se discuten más adelante, es necesario que el modelo del microcontrolador tenga un mínimo de 4 pines para conectarse a la GBA mas los pines necesarios para conectarse al resto de periféricos. Por esta razón se toma como requerimiento que el dispositivo tenga un mínimo de 20 pines. Por otro lado, no es necesario que el PIC funcione a una frecuencia mucho mas alta que la GBA. Con una CPU que funciona a 32 MHz es suficiente e incluye una amplia cantidad de modelos distintos.

Una vez aplicado los filtros, cabe seleccionar el modelo específico para utilizar. Las dos grandes familias de PIC que pasan el filtro son el PIC16F [14] y el PIC24F [15]. El segundo destaca sobre el primero por dos principales factores: modo reposo de bajo consumo y el doble de puertos de comunicación soportados por hardware.

Por tanto, el microcontrolador a utilizar será del conjunto PIC24F. En concreto será el modelo PIC24F16KA101 porque es el único disponible en el laboratorio del ESAII.

La GameBoy Advance

Contexto histórico

La GBA fue la apuesta de Nintendo por una nueva consola portátil. Fabricada desde 2001 hasta 2008, tomó el relevo de la GameBoy (GB) [16] y todas sus vesiones manteniendo compatibilidad con sus juegos. Acogió grandes juegos como Super Mario Advance y Pokemon Rubí/Zafiro. Compitió durante la sexta generación de consolas contra la NeoGeo Pocket Color y otras consolas pero se llevó prácticamente la totalidad del mercado de las consolas portátiles [17].

Nuevas revisiones de esta consola aparecieron a lo largo de su ciclo de vida. La GameBoy Advance SP [18], lanzada en 2003, contaba con un nuevo modelo plegable al estilo de un ordenador portátil además de una batería de litio recargable y una pantalla LCD con iluminación frontal. Este nuevo modelo resolvía muchas de las quejas recibidas por parte de los usuarios sobre el modelo original.

La GameBoy Micro [19], lanzada en 2005, fue la segunda revisión de la consola. Este nuevo modelo, más pequeño y compacto, dejó a un lado la retrocompatibilidad para centrarse en la personalización de la máquina, pues su carcasa podía ser reemplazada con otras de distinto color. La GameBoy Micro, a pesar de su potencia y diseño, no tuvo mucho éxito debido a que fue lanzada muy tarde en el ciclo de vida de la GameBoy Advance y quedó a la sombra de la nueva consola de Nintendo: la Nintendo DS [20].

A lo largo de su ciclo de vida, la GBA recibió diversos periféricos como el adaptador inalámbrico, que permitía contectividad con otras consolas por infrarrojos, o un conector con la GameCube [21], con el que se permitía la comunicación entre la serie de consolas portátiles de Nintendo y su serie de consolas de sobremesa.

Figura 3: Modelo original de la GameBoy Advance

Descripción física

La GBA es un sistema de videojuegos portátil sin almacenamiento permanente ni sistema operativo. El usuario introduce un cartucho en la ranura correspondiente e inicia la máquina. Ésta realiza una comunicación inicial con el cartucho para determinar si se trata de un juego para la GB o la GBA y comienza a ejecutar el programa del cartucho. Existe método alternativo de arranque que pasa por el puerto de comunicaciones. No se cubre en este documento.

Los cartuchos de la GB son de forma cuadrada con 56mm de anchura y 65mm de altura. Adicionalmente tienen una muesca en la parte superior derecha a modo indicativo de la orientación que debe tomar el cartucho a la hora de ser introducido en la ranura de la máquina. Los cartuchos de GBA son de forma rectangular y miden 57mm de anchura y 28mm de altura. La parte superior del cartucho cuenta con una protuberancia de 60mm de anchura y 7mm de altura como guía al estilo de los cartuchos de la GB. Ambos modelos suelen incorporar una pegatina representativa en la parte frontal.

El chip en los cartuchos contiene una memoria, usualmente una ROM, con el programa que la consola ejecutará. Algunos cartuchos incorporan memoria RAM, junto a una batería que la sustente, para guardar datos de la partida. Otros añaden elementos extra a modo de periféricos como un sensor solar, motores de vibración y demás.

Figura 4: Cartucho de GameBoy Color

Figura 5: Cartucho de GameBoy Advance

La comunicación entre la GBA y el cartucho se realiza a través de una región de la memoria de la máquina. Los accesos a direcciones de esta región se corresponden con direcciones del cartucho. La máquina solo permite accesos de lectura de 16 bits. El tamaño de programa de un cartucho viene limitado por el rango de direcciones en esta región y es de un máximo de 32MB. Para acceder a la memoria RAM del cartucho, se define otra región en el espacio de direcciones de la máquina. Esta región tiene un máximo de 64KB y permite accesos de lectura y escritura de 8 bits.

La GameBoy Advance original es de forma rectangular, ligeramente curvada en sus esquinas. En el centro de parte frontal se encuentra la pantalla LCD. A la izquierda de la misma hay una cruceta de control y los botones start y select. En la parte derecha están los botones A y B así como una luz que indica el estado de la batería y un pequeño altavoz. La parte trasera tiene la ranura para los cartuchos y el puerto de comunicación en la parte superior y la cabina de pilas en la inferior.

El diseño de la GameBoy Advance SP es plegable y de forma cuadrada. En su estado desplegado, se distinguen dos partes. La sección superior está ocupada íntegramente por la pantalla. La parte frontal de la sección inferior incluye los botones y un altavoz. El lateral derecho tiene el control de encendido y apagado además de un par de luces indicativas del estado de la batería y carga. El lateral izquierdo tiene el control d volúmen. La parte inferior incluye la ranura para cartuchos y la superior los puertos de comunicación y carga.

La GameBoy Micro es un sistema muy difícil de encontrar hoy en día. Además, ciertas especificaciones técnicas varían respecto a los modelos anteriores. Por ello no se considera a lo largo de este proyecto.

La comunicación entre dos, o más, consolas es posible a través del denominado Cable Link, propietario de Nintendo. Este cable se introduce en los puertos de comunicación de sendos sistemas e implementa un protocolo propietario. Nintendo también lanzó al mercado otro mecanismo de comunicación por infrarrojos.

Figura 6: De izquierda a derecha: GameBoy Advance, GameBoy Advance SP y GameBoy Micro

Además de comunicar varios sistemas, el puerto de comunicaciones sirve para añadir funcionalidades a la máquina en forma de accesorio. Este proyecto tiene como objetivo diseñar un periférico que se comunique con la GBA a través del puerto de comunicaciones.

Descripción del hardware

Procesador

El procesador principal de la GBA es un ARM7-TDMI de 32 bits a 16.8 MHz. Implementa el juego de instrucciones ARMv4 además de un juego de instrucciones de tamaño, 16 bits, y funcionalidad reducida denominado Thumb. Otra característica de este procesador es una unidad de multiplicación rápida. Es uno de los diseños de ARM más extendidos en sistemas empotrados. Por ejemplo, el Nokia 6110.

La GBA cuenta con un coprocesador basado en una versión modificada del Zilog-Z80. Su funcionalidad es ejecutar los juegos de GB. Esto es, cuando el sistema arranca, la máquina lee del cartucho una cabecera que indica si es un juego de GB o de GBA. En el primer caso, el procesador ARM se para y comienza a ejecutar el Z80.

Memoria

La siguiente tabla clasifica rangos de direcciones de memoria en diversas regiones.

Nombre	Inicio	Fin	Tamaño de la región	Tamaño del puerto
System ROM	0x00000000	0x00003FFF	16KB	32bits
EWRAM	0x02000000	0x0203FFFF	256KB	16bits
IWRAM	0x03000000	0x03007FFF	32KB	32bits
IO RAM	0x04000000	0x040003FF	1KB	16bits
PAL RAM	0x05000000	0x050003FF	1KB	16bits
VRAM	0x06000000	0x06017FFF	96KB	16bits
OAM	0x07000000	0x070003FF	1KB	32bits
PAK ROM	0x08000000	variable	máximo de 32MB	16bits
Cart RAM	0x0E000000	variable	máximo de 64KB	8bits

Cuadro 1: Mapa de las regiones de memoria en la GBA

La región de memoria denominada System ROM da acceso a funciones de la BIOS, presentes en el hardware de la máquina. Estas pueden ser rutinas de compresión y descompresión de datos, gestión de interrupciones y demás.

External Work RAM (EWRAM) es la región de memoria más alejada de la CPU. Por tanto, los accesos son lentos. Si se almacena código en esta región, se recomienda que sea de la familia Thumb ya que el puerto de lectura es de 16 bits.

Internal Work RAM (IWRAM) es la región de memoria más cercana a la CPU. Por tanto, los accesos son rápidos. Si el programa utiliza el juego de instrucciones de ARM de 32 bits, se recomienda localizalo en esta región.

La región de memoria denominada IO RAM da acceso a varios registros de control de la máquina. Por ejemplo, configuración del modo de visualización, los canales de audio, transferencias de memoria (DMA) entre otros.

La región de memoria denominada PAL RAM contiene las paletas de colores que se utilizan para pintar por pantalla en ciertas configuraciones.

Video RAM (VRAM) es la región de memoria en la que la máquina interpreta la información a mostrar por pantalla. Existen diversas configuraciones que imponen un orden concreto de los datos en esta región de memoria que se detallan más adelante.

Object Attribute Memory (OAM) es la región de memoria que contiene informaciones sobre objetos, también denominados sprites. Los objetos son entidades gráficas especiales. Su uso se detalla más adelante.

PAK ROM y Cart RAM son los rangos de direcciones que dan acceso a la ROM y la RAM del cartucho respectivamente. El tamaño de estas regiones es variable y depende de la implementación del cartucho en uso.

Herramientas de desarrollo

El código para GBA será en lenguaje C y se utilizará en un editor de texto cualquiera. Existen librerías en C que cubren muchas de las funcionalidades básicas de la consola como pintar sobre la pantalla o llamadas a la BIOS [22]. Ninguna de estas librerías incluye soporte completo para el puerto de comunicación. Por esta razón no se utilizará ninguna librería externa y se desarrollará solo el código necesario para comunicar la consola con el sistema periférico y poder ver los resultados en pantalla.

Los ejecutables se generarán utilizando un compilador específico para la arquitectura de la consola que proviene del conjunto de herramientas de desarrollo devkitARM [23]. Una vez generado el binario se podrá ejecutar sobre el ordenador haciendo uso de un emulador, mgba [24], o bien en la misma GBA.

Para poder ejecutar los programas sobre la consola es necesario utilizar un cartucho modificado o bien mediante un protocolo denominado multiboot. El segundo método hace uso del puerto de comunicaciones, así que queda descartado porque ahí es donde se concetará el sistema periférico. La alternativa es utilizar un cartucho que contiene una ranura para tarjetas microSD y una ROM interna que permite cargar y navegar el sistema de ficheros de la tarjeta [25]. De esta manera se pueden almacenar diversos ejecutables dentro de la microSD y lanzarlos sobre la propia GBA.

El emulador permite ver y analizar toda la memoria de la consola pero no hay ningún mecanismo para emular la comunicación por el puerto serie. Los juegos que implementan este tipo de funcionalidad están parcialmente soportados en el emulador o bien cuentan con un plugin específico para permitir la comunicación entre varias instancias del mismo. Por tanto, todas las pruebas que hagan uso del puerto serie deben hacerse sobre la consola.

Para analizar el comportamiento de los distintos pines del puerto serie será necesario medir su voltaje. Para ello se utilizará un osciloscopio. Esto resultará muy útil a la hora de comprobar la evolución temporal de las distintas señales implicadas en el protocolo de comunicación.

Memoria de video

A pesar de que la funcionalidad principal a implementar es la comunicación por el puerto serie es importante hacer uso de la pantalla para mostrar información tanto al usuario como al desarrollador. Por ello esta sección detalla el funcionamiento, a nivel del programador, de la región de memoria dedicada a la pantalla: la VRAM.

La VRAM es una región de memoria que se extiende el en rango de direcciones `0x06000000-0x06017FFF`. Para poder dibujar cualquier elemento en pantalla basta con escribir dentro de esta región de memoria y la controladora de video se encargará del resto. Este proceso ocurre unas 60 veces por segundo (59.73 Hz).

La controladora tarda un tiempo no negligible para pintar toda la pantalla. A esta ventana de tiempo de le denomina vdraw y dibuja los pixeles por filas comenzando por la equina superior izquierda y terminando en la esquina inferior derecha. Cada vez que la controladora termina de dibujar una fila (240 pixeles), o scanline, pausa el procedimiento un tiempo (equivalente al tiempo necesario para pintar 68 pixeles) antes de comenzar la siguiente. El tiempo entre el fin de una scanline y el comienzo de la siguiente se le denomina hblank.

Una vez la controladora termina de pintar la pantalla entera descansa durante un periodo mas largo de tiempo (83776 ciclos). Esta ventana de tiempo se denomina vblank.

La Figura 7 muestra de forma esquemática las etapas de la controladora de video. La imagen se ha anotado para mostrar el tiempo, en pixeles, de cada etapa y su traducción en ciclos.

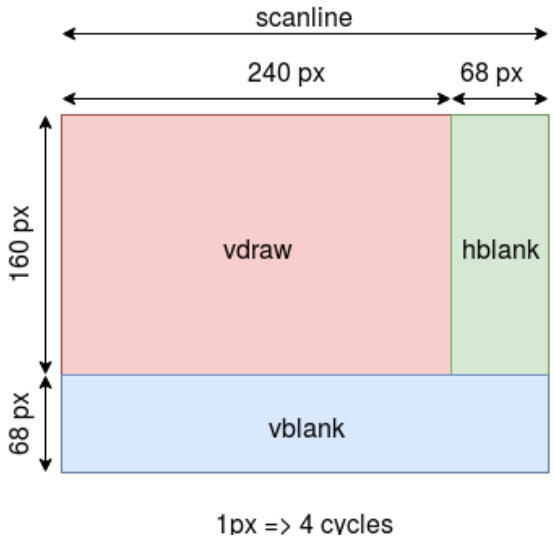

Figura 7: Comportamiento de la controladora de video de la GBA

Es de especial importancia tener en cuenta el comportamiento de la controladora de video porque no hay ningún mecanismo que impida escribir sobre la VRAM mientras la controladora se encuentra en vdraw. Esto quiere decir que podría darse el caso de que se esté pintando un fotograma mientras el programador está escribiendo el siguiente. El resultado final sería una pantalla cuyo contenido muestra parcialmente dos fotogramas distintos.

Por tanto, el tiempo que tiene el programador para escribir sobre la VRAM se encuentra dentro del hblank y el vblank. Debido a que el hblank solo se extiende 272 ciclos por scanline, la cantidad de código que puede ejecutarse es muy reducido. Se suele utilizar esta ventana para implementar efectos gráficos como deformaciones de onda tal y como muestra la Figura 8. La región de vblank es mucho mas extensa temporalmente que el hblank y es donde la mayoría de aplicaciones realizan todo el trabajo relacionado con la memoria de video.

Para sincronizar el flujo del programa con estas regiones de tiempo el programador tiene dos alternativas: hacer una espera activa leyendo una región de memoria concreta hasta determinar si

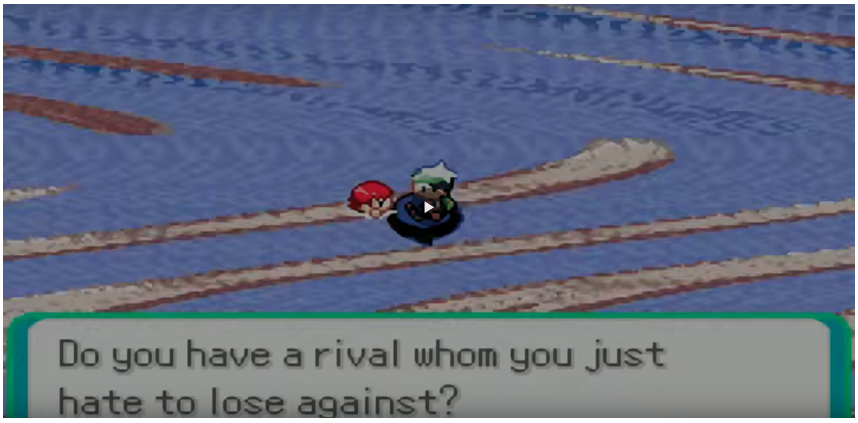

Figura 8: Efecto de deformación de ondas implementado durante el hblank

se encuentra en hblank o vblank; o bien esperar una interrupción hardware que ocurre al comienzo de cada hblank y vblank.

Aplicaciones sencillas pueden hacer uso de la espera activa. Su principal inconveniente es que la CPU sigue trabajando mientras se espera a estar en hblank o vblank y esto incurre en un problema de gasto energético. Es de vital importancia evitar trabajo innecesario del procesador para no gastar la batería de la consola y reducir su autonomía. La espera inactiva a una interrupción es un mecanismo mucho mas sofisticado pero evita el gasto energético de la espera activa.

El programador tiene acceso a un contador en memoria que le permite inferir el estado de la controladora de video. Se trata del `VCOUNT` y se puede definir en C de la siguiente manera:

```
volatile uint16_t* VCOUNT = (uint16_t*)0x04000006;
```

Nótese el uso del atributo `volatile`. En el caso de una espera activa deberá leerse el valor de esta posición de memoria repetidas veces sin escribir nunca en ella. Esto es posible que lleve al compilador a intentar optimizar las lecturas, guardando el valor de la primera lectura en un registro de la CPU y omitiendo las lecturas posteriores. Utilizando la cláusula `volatile` se impide que el compilador realice esta optimización.

Un ejemplo de espera a vblank activa se muestra a continuación:

```
volatile uint16_t* VCOUNT = (uint16_t*)0x04000006;

int main() {
  init(); // Inicializa la aplicacion

  while (1) {
    update(); // Logica de la aplicacion

    while (*VCOUNT > 160) __asm__("nop");
    while (*VCOUNT < 160) __asm__("nop");

    draw(); // Escritura sobre la VRAM
  }

  return 0;
}
```

Nótese que el código incluye dos bucles de espera activa. El segundo, `while (*VCOUNT < 160)`, espera que la controladora esté en estado vblank. Se espera a que el valor del contador `VCOUNT`

sea 160 porque corresponde al número de scanlines que have la controladora. El primer bucle, `while (*VCOUNT > 160)`, sirve a modo de barerra en el caso que los métodos `draw()` y `update()` hayan tardado menos que la espera de la controladora de video y ésta siga en estado vblank. Si no se incluye esta guardia es posible que se alcance el segundo blucle mas de una vez durante un mismo fotograma. Este comportamiento se muestra de forma esquemática en la Figura 9.

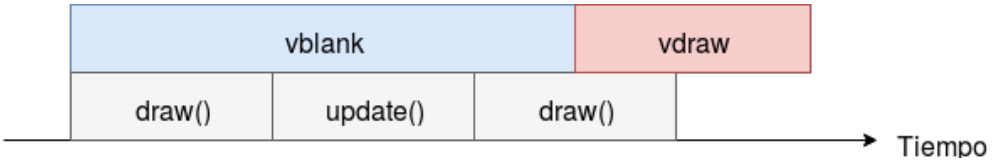

Figura 9: Dos fotogramas pintados durante el mismo vdraw

El mismo ejemplo puede modificarse para hacer uso de el sistema de interrupciones de la consola y evitar la espera activa:

```
int main() {
  init(); // Inicializa la aplicacion

  irq_init(0x0000); // Inicializa las interrupciones
  irq_add(II_VBLANK, NULL); // Activa la interrupcion del vblank

  while (1) {
    update(); // // Logica de la aplicacion

    bios_vblank_wait(); // Pausa la CPU hasta que ocurra una interrupcion de vblank

    draw(); // Escritura sobre la VRAM
  }

  return 0;
}
```

Esta vez la CPU entrará en un modo de bajo consumo hasta recibir la interrupción que la controladora lanza al entrar en la región de vblank. Nótese que ya no es necesario incluir una guardia que evite pintar dos fotogramas en un mismo vblank porque la llamada `bios_vblank_wait();` pausa la CPU aun estando dentro de vblank.

El detalle de la implementación de las interrupciones se detalla en la Sección 4.7.

Puerto de comunicación

La GBA tiene un puerto de comunicación serie. El factor forma del mismo es propietario por lo que obliga a utilizar un cable fabricado específicamente para la GBA. Este cable recibe el nombre de Cable Link y fue utilizado por varios juegos comercializados para conectar de dos a cuatro consolas y permitir una experiencia multijugador. El puerto tiene seis pines descritos en la Tabla **??**.

Nombre	**Nomenclatura UART**
V_{cc}	-
SO	T_x
SI	R_x
SD	RTS
SC	CTS
GND	-

La consola soporta varios modos de comunicación por el puerto serie. Tres de ellos son propietarios: Normal, Multi-Player y JOY-BUS. Esto quiere decir que es necesario que el dispositivo

conectado a la GBA lo utilice también pero no será posible a no ser que se trate de otra GBA.

Por suerte Nintendo incluyó la posibilidad de utilizar el puerto serie mediante el protocolo estándard UART. De esta manera se puede conectar la consola con cualquier otro dispositivo que utilice este mismo protocolo.

Por último existe la posibilidad de utilizar el puerto serie como un conjunto de pines genérico escribiendo y leyendo cada uno de manera separada. Este modo se denomina General Purpose.

Sistema de interrupciones

La GBA cuenta con un sistema de interrupciones hardware que permite controlar de forma eficaz el flujo del programa. Cuando ocurre una interrupción la CPU deja de ejecutar el código en el que se encontraba y pasa a servir la interrupción mediante la ejecución de la denominada subrutina de servicio. Las interrupciones soportadas por la GBA se listan en la Tabla 2.

Nombre	**Detalle**
Vblank	Cuando la controladora de video entra en estado vblank
Hblank	Cuando la controladora de video entra en estado hblank
Vcount	Cuando el contador `VCOUNT` alcanza un valor establecido por el programador
TimerX	Cuando el contador del `TIMER_X` hace overflow
Comm	Depende del modo del puerto serie
Dma	Cuando la controladora de DMA concluye una transferencia
Keypad	Cuando el input definido por el programador es activado por el jugador
Gamepack	Cuando el usuario quita el cartucho de la ranura

Cuadro 2: Listado de interrupciones hardware de la GBA

Para activar las interrupciones en general el programador debe escribir el valor `0x1` sobre el registro de memoria `IME`.

```
volatile uint32_t* IME = (uint32_t*)0x04000208;
```

Para activar una interrupción en concreto el programador debe activar el bit correspondiente en el registro de memoria `IE`.

```
/**
  Bit   Expl.
  0     LCD V-Blank                    (0=Disable)
  1     LCD H-Blank                    (etc.)
  2     LCD V-Counter Match            (etc.)
  3     Timer 0 Overflow               (etc.)
  4     Timer 1 Overflow               (etc.)
  5     Timer 2 Overflow               (etc.)
  6     Timer 3 Overflow               (etc.)
  7     Serial Communication           (etc.)
  8     DMA 0                          (etc.)
  9     DMA 1                          (etc.)
  10    DMA 2                          (etc.)
  11    DMA 3                          (etc.)
  12    Keypad                         (etc.)
  13    Game Pak (external IRQ source) (etc.)
  14-15 Not used
*/
volatile uint16_t* IE = (uint16_t*)0x04000200;
```

Cuando una interrupción ocurre el bit correspondiente se activa en el registro de memoria `IF`. El programador deberá indicar que ya ha servido esta interrupción escribiendo un `1` sobre el bit. La razón por la que debe escribirse un `1` y no un `0` no se esclarecen en la documentación.

```
/**
  Bit   Expl.
  0     LCD V-Blank                    (1=Request Interrupt)
  1     LCD H-Blank                    (etc.)
  2     LCD V-Counter Match            (etc.)
  3     Timer 0 Overflow               (etc.)
  4     Timer 1 Overflow               (etc.)
  5     Timer 2 Overflow               (etc.)
  6     Timer 3 Overflow               (etc.)
  7     Serial Communication           (etc.)
  8     DMA 0                          (etc.)
  9     DMA 1                          (etc.)
  10    DMA 2                          (etc.)
  11    DMA 3                          (etc.)
  12    Keypad                         (etc.)
  13    Game Pak (external IRQ source) (etc.)
  14-15 Not used
*/
volatile uint16_t* IF = (uint16_t*)0x04000202;
```

Debe existir una rutina de servicio de interrupciones para que éstas funcionen. La dirección en memoria de esta subrutina debe especificarse en el registro de memoria `ISR_MAIN`. En el caso de que la aplicación solo haga uso de un tipo de interrupción concreto, la subrutina de servicio puede ser muy sencilla. Si el programador planea utilizar mas de un tipo de interrupción será necesario gestionar cuál de ellas se sirve al entrar en la subrutina apuntada por `ISR_MAIN`.

Un primer acercamiento sería construir un control del tipo `if-then-else` y ejecutar el bloque de código correspondiente a cada tipo de interrupción. Esto sería altamente ineficiente y tendría un resultado nefasto en la aplicación, pues las interrupciones deben servirse en el menor tiempo posible para devolver el control al flujo principal del programa.

La solución es crear una tabla con tantos registros como tipos de interrupción se quiera soportar y donde cada registro contenga la dirección de memoria en la que se encuentra la rutina de servicio para un tipo de interrupción en concreto.

```
typedef enum eIrqIndex {
  II_VBLANK=0,II_HBLANK,  II_VCOUNT,  II_TIMER0,
  II_TIMER1,  II_TIMER2,  II_TIMER3,  II_SERIAL,
  II_DMA0,  II_DMA1,  II_DMA2,  II_DMA3,
  II_KEYPAD,  II_GAMEPAK, II_MAX
} eIrqIndex;

typedef struct IRQ_REC   {
  uint32_t flag;
  fnptr isr;
} IRQ_REC;

IRQ_REC __isr_table[II_MAX+1];
```

Con todas estas consideraciones se puede escribir la rutina principal de servicio a la interrupciones. Es importante que ésta sea lo mas rápida posible por lo que es aconsejable que se encentre en la región de memoria IWRAM, la mas cercana físicamente a la CPU. El código de la IWRAM debe ser del tipo THUMB para ocupar lo menos posible. El compilador no es capaz de generar subrutinas con este subconjunto del juego de instrucciones de la CPU por lo que el código debe escribirse en ensamblador. Para mayor legibilidad, a continuación se muestra cómo sería el código equivalente en C.

```
void isr_master_c() {
  uint32_t ie   = *IE;
  uint32_t ieif = ie & (*IF);
  IRQ_REC* pir;

  // Escribir 1 en IF para todas las interrupciones activas
  *IF = ieif;

  // Recorrer la tabla de los tipos de interrupcion soportados
  // para encontrar si la interrupcion recibida puede servirse
  for(pir =  __isr_table; pir->flag !=0; pir++)
    if(pir->flag & ieif)
      break;

  // Si no hay subrutina de servicio para este tipo de interrupcion
  // salir de isr_master
  if(pir->flag == 0 || pir->isr == NULL)
    return;

  // Si hay subrutina de servicio para este tipo de interrupcion
  // Desactivar interrupciones mientras se sirve la actual
  uuint32_t ime = *IME;
  *IME = 0;
  *IE &= ~ieif;

// Ejecutar la subrutina de servicio para la interrupcion
  pir->isr();

  // Activar de nuevo las interrupciones
  *IE  = ie;
  *IME = ime;
}
```

Para los propósitos de este proyecto solo son de interés la interrupción Vblank y los contadores de tiempo `TIMER_X`. El uso del Vcount se detalla en la Sección 4.5. Los contadores de tiempo se utilizan para sincronizar aplicaciones que dependen del tiempo.

La GBA cuenta con 4 contadores de tiempo y cada uno de ellos se mapea en memoria con dos estructuras de datos: una es el propio contador `TMXCNT_L` y la otra es el control del mismo `TMXCNT_H`. Una posible definición en C de estos contadores sería:

```
volatile uint16_t* TM0CNT_L = (uint16_t*)0x04000100;
volatile uint16_t* TM1CNT_L = (uint16_t*)0x04000104;
volatile uint16_t* TM2CNT_L = (uint16_t*)0x04000108;
volatile uint16_t* TM3CNT_L = (uint16_t*)0x0400010C;

/**
  Bit   Expl.
  0-1   Prescaler Selection (0=F/1, 1=F/64, 2=F/256, 3=F/1024)
  2     Count-up Timing   (0=Normal, 1=Cascade)
  3-5   Not used
  6     Timer IRQ Enable  (0=Disable, 1=IRQ on Timer overflow)
  7     Timer Start/Stop  (0=Stop, 1=Operate)
  8-15  Not used

  F = System Clock (16.78MHz).
*/
volatile uint16_t* TM0CNT_H = (uint16_t*)0x04000102;
volatile uint16_t* TM1CNT_H = (uint16_t*)0x04000106;
volatile uint16_t* TM2CNT_H = (uint16_t*)0x0400010A;
volatile uint16_t* TM3CNT_H = (uint16_t*)0x0400010E;
```

El modo de uso de un timer pasa por configurarlo utilizando el `TMXCNT_H` correspondiente, introduciendo el valor inicial del contador en `TMXCNT_L` y activar el timer escribiendo sobre el bit 7 de la palabra de control. Desde ese momento el contador se incrementará con la frecuencia seleccionada en los bits 0-1 y emitirá una interrupción cuando el conador `TMXCNT_L` haga overflow.

El contador de tiempo es de 32bits por lo que hará overflow cuando pase el valor `0xFFFF` (65535 en decimal). Los incrementos se hacen a la frecuencia de la CPU (16.78 MHz) escalada por el factor $PRESCALER$. A continuación se define una expresión genérica para calcular el tiempo que tardará el contador en hacer overflow:

$$T = \frac{65535 - TMXCNT\_L}{f} \cdot PRESCALER$$

El ejemplo siguiente inicializa el Timer0 para emitir una interrupción después de unos 323 milisegundos de ser activado.

```
*TM0CNT_L = 0x1234; // 4660
*TM0CNT_H = 0x0001  // Prescaler 64
          | 0x0040  // Interrupcion al hacer overflow
          | 0x0080; // Start
```

Aplicando la expresión anterior:

$$T = \frac{65535 - 4660}{16{,}78 * 10^{-6}} \cdot 64 \approx 232{,}18ms$$

Un contador puede configurarse para tardar un máximo de:

$$T = \frac{65535 - 0}{16{,}78 * 10^{-6}} \cdot 1024 \approx 4s$$

Si el contexto de la aplicación requiere un timer mas extenso puede utilizar la funcionalidad de cascading que proporciona la GBA. Con esto el timer X se configurará a una frecuencia dada e incrementará el timer $X-1$ cuando haga overflow. Esto funciona en todos los timers de la consola salvo el `TIMER_0`.

El servicio de la interrupción del timer puede ser a través de una subrutina o no. El mecanismo para especificar la subrutina de servicio en los timers es el mismo que para el resto de interrupciones hardware de la consola.

Con esto se puede escribir un conjunto de funciones que ayuden al programador a utilizar los timers de la consola.

```
#define TIMER_PRESCALER_1    0
#define TIMER_PRESCALER_64   1
#define TIMER_PRESCALER_256  2
#define TIMER_PRESCALER_1024 3

#define TIMER_CASCADE 0x0004
#define TIMER_ENABLE  0x0080

void timer0_init(uint16_t count, int prescaler, fnptr irq) {
  irq_add(II_TIMER0, irq);
  irq_enable(II_TIMER0);

  *TM0CNT_L = count;
  *TM0CNT_H = TIMER_ENABLE | prescaler | 0x0040;
}

void timer1_init(uint16_t count, int prescaler, fnptr irq) {
  irq_add(II_TIMER1, irq);
  irq_enable(II_TIMER1);

  *TM1CNT_L = count;
  *TM1CNT_H = TIMER_ENABLE | prescaler | 0x0040;
}

void timer2_init(uint16_t count, int prescaler, fnptr irq) {
  irq_add(II_TIMER2, irq);
  irq_enable(II_TIMER2);

  *TM2CNT_L = count;
  *TM2CNT_H = TIMER_ENABLE | prescaler | 0x0040;
}

void timer3_init(uint16_t count, int prescaler, fnptr irq) {
  irq_add(II_TIMER3, irq);
  irq_enable(II_TIMER3);

  *TM3CNT_L = count;
  *TM3CNT_H = TIMER_ENABLE | prescaler | 0x0040;
}

void timer0_disable() { irq_disable(II_TIMER0); }
void timer1_disable() { irq_disable(II_TIMER1); }
void timer2_disable() { irq_disable(II_TIMER2); }
void timer3_disable() { irq_disable(II_TIMER3); }
```

El PIC24F16KA101

Descripción del hardware

El PIC24F16KA101 (PIC) es un microcontrolador de la familia PIC24 manufacturado por la empresa Michrochip Technology. Se trata de una familia de microcontroladores ampliamente usados en el mercado. A lo largo del grado, los alumnos de la FIB tienen ocasión de trabajar con ellos en un par de asignaturas.

El modelo utilizado en este proyecto es un microcontrolador de 16 bits con especial énfasis en bajo consumo y amplio soporte para conexión con periféricos. No olvidemos que este dispositivo deberá conectarse y ser alimentado por la GBA por lo que el enfoque de bajo consumo es de gran interés. Por otra parte, tener soporte para distintos protocolos de comunicación con periféricos ampliará la capacidad del microcontrolador para actuar como traductor entre la consola y el periférico en sí. La Figura 10 muestra el modelo del PIC utilizado en este proyecto.

Figura 10: Encapsulado del PIC24F16KA101

El PIC tiene una forma rectangular y cuenta con un total 20 pines distribuidos a lo largo de los laterales. Es capaz de almacenar hasta 8KB de memoria de programa y 1.5KB de datos. Tiene también tres timers de 16 bits con un funcionamiento similar al de la GBA. La Figura 11 muestra de manera esquemática el PIC con cada uno de sus pines etiquetados de acuerdo al fabricante.

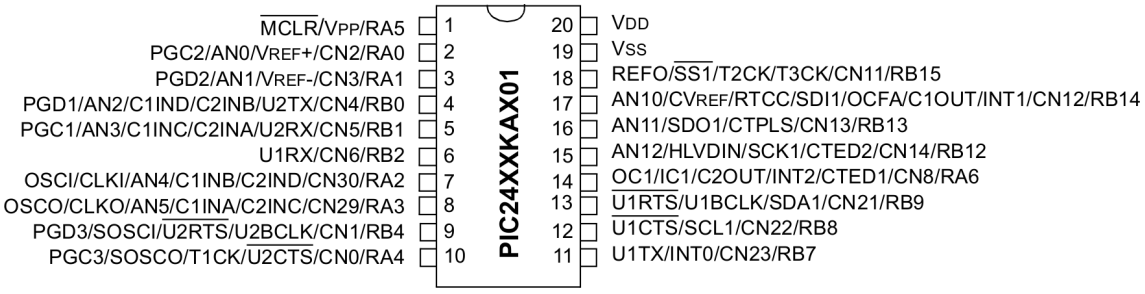

Figura 11: Pinout del PIC24F16KA101

Herramientas de desarrollo

Los programas para el PIC se escribirán en C utilizando el entorno de desarrollo del fabricante: MPLAB. Éste incluye un editor y un compilador gratuito, xc compiler, que no implementa optimizaciones. El entorno también cuenta con librarías de desarrollo con las definiciones necesarias para facilitar la programación del dispositivo.

Una vez escrito el programa deberá transferirse sobre el propio microcontrolador. Esto se hará utilizando el programador del fabricante, el ICD3. Este dispositivo no es gratuito pero me ha sido cedido temporalmente por el departamento del ESAII. El ICD3 tiene otras funcionalidades además

de programador pero no se consideran para este proyecto. La Figura 12 muestra el programador ICD3.

Figura 12: Programador ICD3

Puertos de comunicación

Los puertos de comunicación soportados de manera nativa en el microcontrolador son UART (2 instancias), SPI e I2C. Todos ellos son protocolos estándar y se utilizan en la mayoría de los sensores y microcontroladores del mercado. Para los propósitos del proyecto se planificó utilizar el puerto de comunicación UART para conectar el PIC y la GBA. No obstante, como se discute en la Sección 6, el protocolo que se utiliza para comunicar las dos máquinas no es estándard. Por esta razón no se utilizarán los puertos de comunicación del PIC.

Sistema de interrupciones

De manera similar a la GBA, el PIC tiene un sistema de interrupciones con el que controlar de manera precisa el flujo del programa. El principal uso de las interrupciones durante este proyecto será el de los timers, que funcionan de manera muy similar a los timers de la GBA.

El PIC tiene tres timers distintos con contadores de 16 bits de los cuales el par Timer2-Timer3 puede ser combinado en un solo contador de 32 bits. Para los propósitos de este proyecto solo se utiliza el Timer1.

Para configurar el Timer1 basta con seguir los pasos siguientes:

1. Poner a 1 el bit TON del registro T1CON
2. Seleccionar el factor de prescaler usando los bits TCKPS del registro T1CON
3. Cargar el valor inicial del contador en el registro PR1

El procedimiento es análogo al descrito en la GBA. La única diferencia es que el propio entorno del MPLAB proporciona las definiciones de los registros necesarios.

Desarrollo de sistema de comunicación entre las máquinas

UART

El protocolo de comunicación UART es un estándar en comunicación serie. Por suerte tanto la GBA como el PIC tienen soporte hardware para este tipo de comunicación. En el protocolo entran en juego las señales siguientes: Tx, Rx, RTS y CTS.

Por desgracia, es muy difícil encontrar un cable link oficial para la GBA. Conseguí comprar uno pero resultó ser defectuoso en que uno de los pines (SI) no funcionaba. No pude comprar otro de otro proveedor por lo que tuve que conformarme con un cable link no oficial.

Por si no fuera poco, el cable no oficial no tiene todos los pines del cable oficial, pues no son necesarios para la mayoría de juegos comercializados. En concreto tiene los pines: Vcc, SI, SD, SC y GND. La consecuencia de esto es que no se puede utilizar el módulo UART integrado en la GBA.

Las Figuras 13 y 14 muestran la diferencia de pines entre el cable oficial y el no oficial.

Figura 13: Cable link oficial con 6 pines

Figura 14: Cable link no oficial con 4 pines

Valoración de alternativas

El resto de protocolos que soporta la consola son propietarios del fabricante. Esto implica que están específicamente diseñados para funcionar con otras GBAs o productos del comerciante, no con un microcontrolador cualquiera.

La primera alternativa a utilizar el protocolo UART sería implementar, emulando por software, sobre el PIC uno de los protocolos propietarios de la GBA. Esto es difícil porque la documentación de los protocolos es privada y no hay documentación oficial sobre ellos. Algunos desarrolladores independientes han hecho ingeniería inversa para inferir el comportamiento del protocolo pero no aseguran un correcto funcionamiento.

La alternativa que este proyecto propone es utilizar el puerto serie con el modo de propósito general y diseñar un protocolo alternativo al UART para que la GBA y el PIC se entiendan.

Descripción de un protocolo alternativo

El nuevo protocolo podrá hacer uso de los únicos pines disponibles en el cable link no oficial del que dispongo: SI, SD y SC. La comunicación será bidireccional. Esto es, tanto la GBA como el PIC pueden iniciar la comunicación y ambas máquinas pueden enviar datos. Se enviará una palabra de 8 bits (1 byte) de información por cada transacción. El protocolo puede extenderse para permitir transacciones mas largas pero, por simplicidad, este proyecto solo implementa comunicaciones de 8 bits.

El pin SI corresponderá a la salida de la GBA y se conectará con la entrada del PIC. Por este cable se enviarán los datos desde la consola hacia el microcontrolador. El pin SO es simétrico al pin SI. Por último, la GBA enviará una señal periódica por el pin SC para que funcione a modo de reloj de sincronización durante las transacciones.

Las señales de datos restan a cero lógico mientras los agentes esperan a enviar y/o recibir datos. Un agente puede iniciar una comunicación introduciendo un uno lógico en su señal de salida de datos. Este cambio lo denominamos start bit y marca el inicio del procedimiento. Si el agente que comienza la comunicación es la GBA ésta comienza a generar una señal de reloj por el pin SC. En caso que la comunicación sea iniciada por el PIC la señal de reloj comenzará cuando la GBA lea el start bit por su pin de entrada de datos.

Después del start bit se enviará el byte de datos. Un bit por cada flanco ascendiente de reloj comenzando por el bit de menor peso y concluyendo con el de mayor. Una vez transmitidos todos los bits la señal de datos del agente que transmite debe volver a cero. Cuando el agente receptor lea el último bit de la transacción deberá enviar un uno lógico por su señal de salida de datos en el siguiente flanco ascendiente de reloj y volver a dejar la señal a cero al próximo ciclo. Este último pulso se denomina ack y marca el final de la transacción.

Nótese que el protocolo no incluye ningún mecanismo de detección ni de corrección de errores. Si uno de los agentes se desconecta durante una transmisión el otro no podrá percatarse de ello. Si la información que el receptor lee es errónea debido al ruido del canal no habrá ninguna forma de detectarlo ni de corregirlo. El único mecanismo de detección que implementa el protocolo es el ack, que sirve únicamente para indicar al transmisor que el receptor ha leído todos los datos enviados. Si el programador quiere, puede implementar algún otro mecanismo para actuar en caso que no reciba la señal de ack.

La Figura15 muestra la evolución de las señales a lo largo del tiempo durante la transmisión de un byte desde la GBA hacia el PIC.

Nótese que el protocolo no define una frecuencia de reloj determinada. Es mas, el reloj puede ser taquicárdico y no afectar al comportamiento del protocolo siempre y cuando ambos agentes tengan tiempo de leer y escribir por los pines concretos dentro de la ventana de tiempo de un ciclo. Por esto mismo la GBA implementará la frecuencia de reloj de 1Hz. Este valor corresponde con un timer con el contador inicializado a `0x8000` y un escalado de factor 256.

$$T = \frac{65535 - 32768}{16{,}78} \cdot 256 \cdot 10^{-6} = 0{,}4999s \approx 0{,}5s$$

El tiempo entre dos flancos de reloj será de 0.5s lo que quiere decir que el tiempo entre dos ciclos de reloj (dos flancos ascendientes) será de 1s, lo que equivale a un reloj de frecuencia 1Hz.

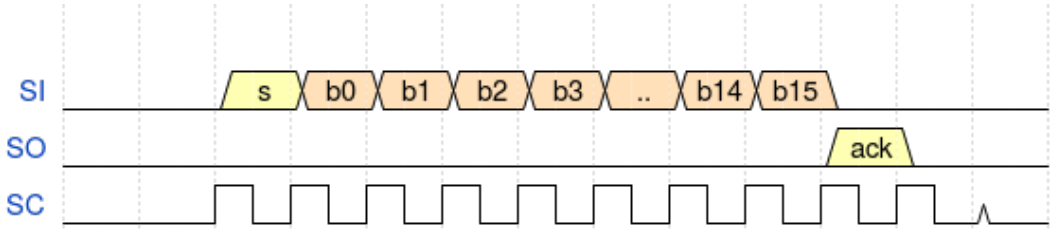

Figura 15: Envío de un byte desde la GBA hacia el PIC

Implementación sobre la GBA

Para seleccionar el modo de propósito general basta con escribir sobre el registro en memoria `SIOCNT` el valor `0x8000`.

```
/**
  Bit   Expl.
  0     Shift Clock (SC)        (0=External, 1=Internal)
  1     Internal Shift Clock    (0=256KHz, 1=2MHz)
  2     SI State (opponents SO) (0=Low, 1=High/None) --- (Read Only)
  3     SO during inactivity    (0=Low, 1=High) (applied ONLY when Bit7=0)
  4-6   Not used                (Read only, always 0 ?)
  7     Start Bit               (0=Inactive/Ready, 1=Start/Active)
  8-11  Not used                (R/W, should be 0)
  12    Transfer Length         (0=8bit, 1=32bit)
  13    Must be "0" for Normal Mode
  14    IRQ Enable              (0=Disable, 1=Want IRQ upon completion)
  15    Not used                (Read only, always 0)
*/
volatile uint16_t* SIOCNT = (uint16_t*)0x04000128;
```

A partir de este momento se puede utilizar el registro en memoria `RCNT` para configurar la direccionalidad de los pines y leer/escribir sobre ellos.

```
/**
  Bit   Expl.
  0     SC Data Bit         (0=Low, 1=High)
  1     SD Data Bit         (0=Low, 1=High)
  2     SI Data Bit         (0=Low, 1=High)
  3     SO Data Bit         (0=Low, 1=High)
  4     SC Direction        (0=Input, 1=Output)
  5     SD Direction        (0=Input, 1=Output)
  6     SI Direction        (0=Input, 1=Output, but see below)
  7     SO Direction        (0=Input, 1=Output)
  8     SI Interrupt Enable (0=Disable, 1=Enable)
  9-13  Not used
  14    Must be "0" for General-Purpose Mode
  15    Must be "1" for General-Purpose or JOYBUS Mode

*/
volatile uint16_t* RCNT = (uint16_t*)0x04000134;
```

Con esto se puede escribir una colección de funciones para facilitar al programador el uso del puerto serie.

```
inline void sio_init() { *SIOCNT = 0x8000; }

inline void sio_set_sc_direction(uint16_t direction) {
  *RCNT = (*RCNT & 0xffef) | (direction << 4);
}
inline void sio_set_sd_direction(uint16_t direction) {
  *RCNT = (*RCNT & 0xffdf) | (direction << 5);
}
inline void sio_set_si_direction(uint16_t direction) {
  *RCNT = (*RCNT & 0xffbf) | (direction << 6);
}
inline void sio_set_so_direction(uint16_t direction) {
  *RCNT = (*RCNT & 0xff7f) | (direction << 7);
}

inline uint16_t sio_get_sc_direction() { return ((*RCNT >> 4) & 0x0001); }
inline uint16_t sio_get_sd_direction() { return ((*RCNT >> 5) & 0x0001); }
inline uint16_t sio_get_si_direction() { return ((*RCNT >> 6) & 0x0001); }
inline uint16_t sio_get_so_direction() { return ((*RCNT >> 7) & 0x0001); }

inline void sio_set_sc_value(uint16_t value) {
  *RCNT = (*RCNT & 0xfffe) | (value << 0);
}
inline void sio_set_sd_value(uint16_t value) {
  *RCNT = (*RCNT & 0xfffd) | (value << 1);
}
inline void sio_set_si_value(uint16_t value) {
  *RCNT = (*RCNT & 0xfffb) | (value << 2);
}
inline void sio_set_so_value(uint16_t value) {
  *RCNT = (*RCNT & 0xfff7) | (value << 3);
}

inline void sio_toggle_sc_value() {
  (sio_get_sc_value() == SIO_HIGH)? sio_set_sc_value(SIO_LOW)
                                  : sio_set_sc_value(SIO_HIGH);
}
inline void sio_toggle_sd_value() {
  (sio_get_sd_value() == SIO_HIGH)? sio_set_sd_value(SIO_LOW)
                                  : sio_set_sd_value(SIO_HIGH);
}
inline void sio_toggle_si_value() {
  (sio_get_si_value() == SIO_HIGH)? sio_set_si_value(SIO_LOW)
                                  : sio_set_si_value(SIO_HIGH);
}
inline void sio_toggle_so_value() {
  (sio_get_so_value() == SIO_HIGH)? sio_set_so_value(SIO_LOW)
                                  : sio_set_so_value(SIO_HIGH);
}

inline uint16_t sio_get_sc_value() { return ((*RCNT >> 0) & 0x0001); }
inline uint16_t sio_get_sd_value() { return ((*RCNT >> 1) & 0x0001); }
inline uint16_t sio_get_si_value() { return ((*RCNT >> 2) & 0x0001); }
inline uint16_t sio_get_so_value() { return ((*RCNT >> 3) & 0x0001); }
```

Implementación sobre el PIC

Para el caso del PIC, leer y escribir sobre los pines es tarea sencilla. El entorno de programación de MPLAB ya contiene las definiciones necesarias para poder acceder de forma cómoda a los pines.

El modelo de PIC utilizado agrupa los pines de propósito general en dos grandes grupos: `RA` y `RB`. Cada uno de los grupos engloba un número determinado de pines y los numera comenzando por el 0, `RA0`, y terminando por el 15, `RA15`. El detalle de la numeración y cómo quedan distribuidos físicamente a lo largo del dispositivo varía según el modelo.

Los pines pueden configurarse como entradas o salidas. Esto se hace mediante los registros denominados `TRIS` y hay uno de estos por cada puerto. En nuestro caso: `TRISA` y `TRISB`. Cada bit `x` de este registro indica la direccionalidad del pin `RAx` o `RBx`. El fabricante utiliza el siguiente nemotécnico para codificar si los pines son entrada o salida:

- Entrada (Input) codificado como `1`. $\rightarrow \quad I \sim 1$
- Salida (Output) codificado como `0`. $\rightarrow \quad O \sim 0$

Por ejemplo, para configurar el puerto RA con los pines `RA0` y `RA1` como entrada y el pin `RA2` como salida, el registro `TRISA` debe valer: `0bxxxxxxxxxxxxx011` en notación binaria(donde `x` denota que el valor del bit es indiferente), o bien `0x0003` en notación hexadecimal.

Una vez configurada la direccionalidad del puerto, basta con utilizar el registro `PORTA` o `PORTB` para leer o escribir sobre los pines. De manera similar al `TRIS`, `PORT` mapea cada bit del registro a un pin del puerto.
El entorno de MPLAB no solo define los registros `TRIS` y `PORT` sino que además lo hace con una estructura de datos que permite acceder a cada bit individualmente con la notación de C de forma cómoda.

```
// Definicion de la direccionalidad con una sola palabra
TRISA = 0x0003;

// Definicion de la direccionalidad pin a pin
TRISAbits.RA0 = 1;
TRISAbits.RA1 = 1;
TRISAbits.RA2 = 0;

/*********************************/

// Lectura de los bits manipulando una sola palabra
SI = (PORTA >> 0) & 0x0001;
SC = (PORTA >> 1) & 0x0001;

// Lectura de los bits pin a pin
SI = PORTAbits.RA0;
SC = PORTAbits.RA1;

/*********************************/

// Escritura de los bits manipulando una sola palabra
PORTA = (SO << 2) & 0x0004;

// Escritura de los bits pin a pin
PORTAbits.RA2 = SO;
```

Resultados

La Figura 16 muestra el estado final del sistema de conexión entre la GBA y el PIC sobre una protoboard. A la izquierda de la imagen se encuentra la consola que he utilizado durante el proyecto: una GameBoy Advance SP. A la derecha se puede ver la protoboard con el PIC en el centro. El cable ancho y negro del cual salen cuatro cables mas pequeños es el cable link. El otro extremo del cable se conecta a la parte posterior de la consola.

En la parte trasera de la placa se ve un grupo de cables (dos verdes, uno rojo y uno negro) que no están conectados al sistema principal. Esa región de la placa se ha destinado al programador del PIC. Cada vez que una nueva versión del código debía transferirse al microcontrolador, el programador se conectaba ahí.

Cabe decir que la GBA no alimenta al PIC en este prototipo. Por simplicidad se ha utilizado una fuente de alimentación con corriente continua a lo largo del proyecto. Las dos columnas de puntos al extremo izquierdo de la placa tienen el voltaje de referencia y masa para que el PIC se alimente.

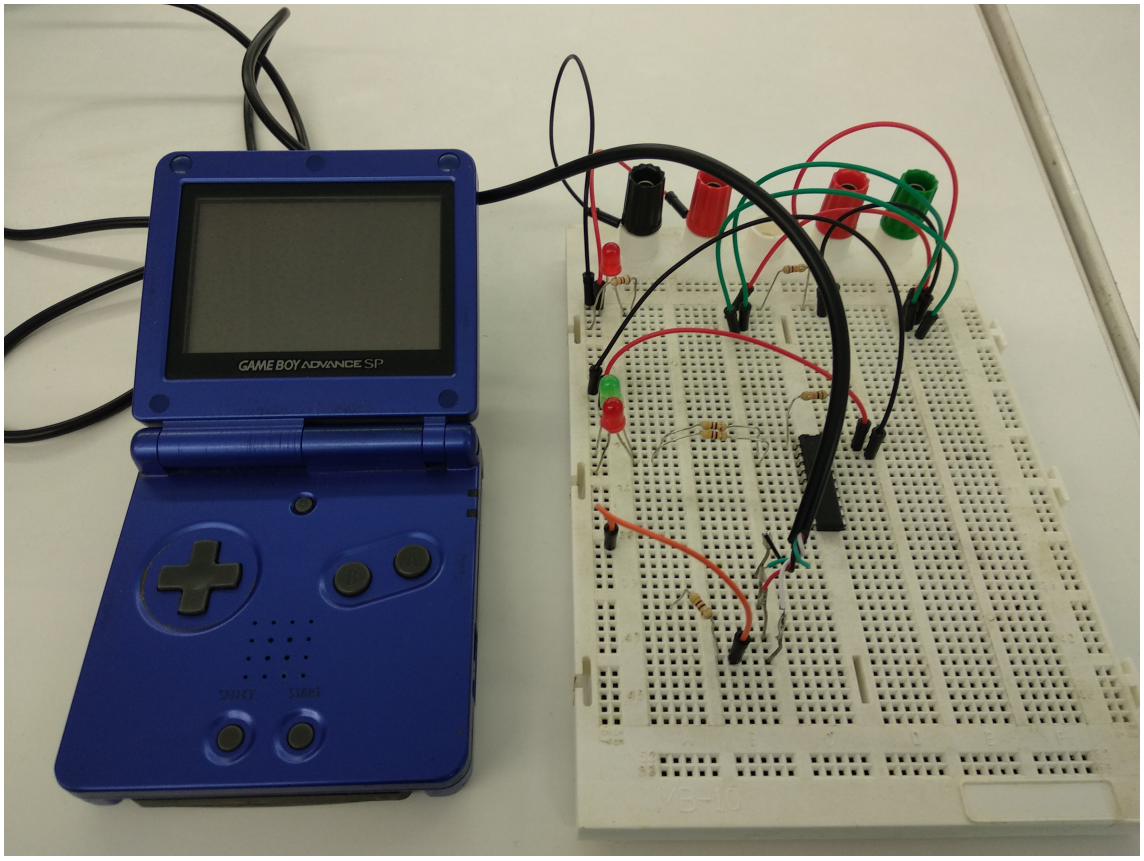

Figura 16: Sistema sobre la protoboard

La Figura 17 es una fotografía de la pantalla del osciloscopio durante la comunicación de la GBA al PIC. La señal de color amarillo es el canal de datos SD mientras que la señal de color rojo el reloj de la comunicación SC. La imagen muestra el final de una comunicación seguida de una pausa y el inicio de otra comunicación.

Nótese que la señal amarilla toma el valor de 1 lógico al inicio de la comunicación (start bit) y procede a enviar uno por uno los bits de la palabra: `00001100` que corresponde con el valor hexadecimal `0x0C` o bien con el valor decimal 12.

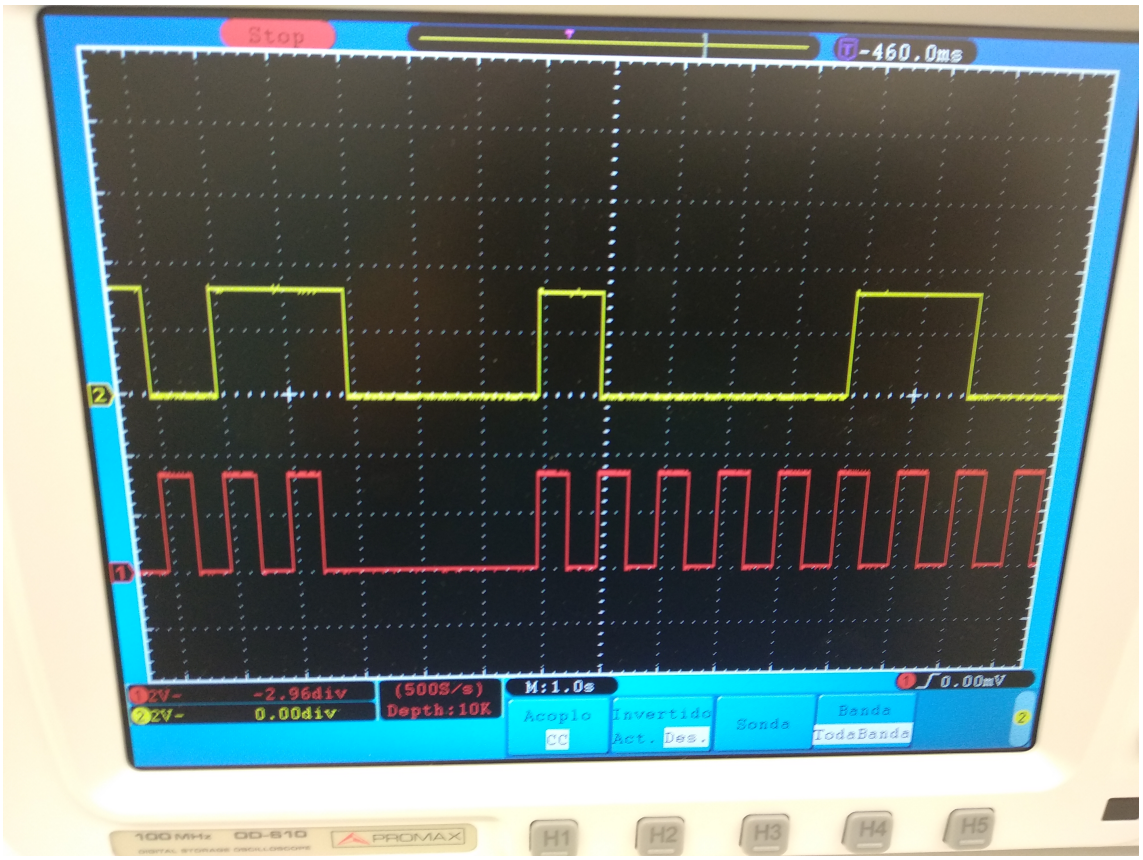

Figura 17: Envío de un byte desde la GBA hacia el PIC

Caso de uso: Acelerómetro

Una vez implementada la comunicación entre el PIC y la GBA se ha decidido probar el sistema con un periférico concreto: El Acelerómetro ADXL335 [26]. Se trata de un dispositivo que calcula la fuerza de la aceleración en tres ejes: X, Y y Z. El motivo por el que se ha elegido este dispositivo es porque hay bastante información en internet sobre él y cómo conectarlo con un modelo de PIC. La Figura 18 muestra el acelerómetro.

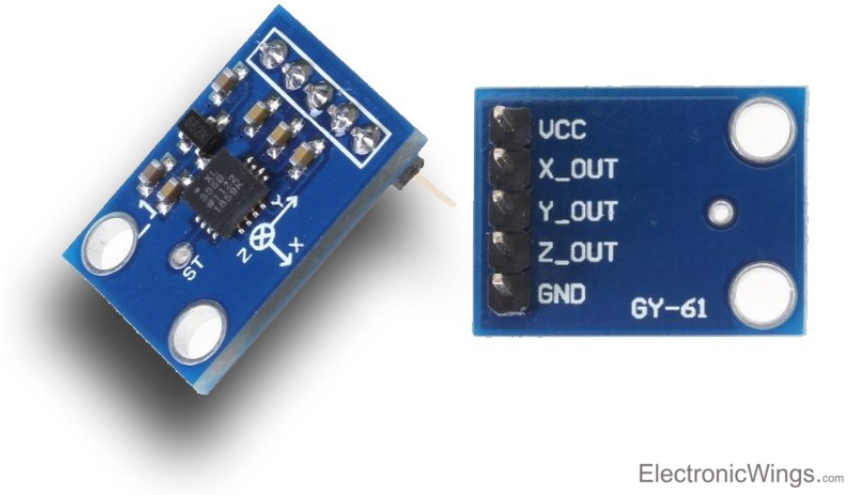

Figura 18: Acelerómetro ADXL335

El ADXL335 envía los datos al PIC por una interfaz analógica. El PIC24F16KA101 tiene suficientes canales analógicos para leer los tres ejes del acelerómetro. Siguiendo la guía [27], el cálculo ha realizar para convertir el valor leído a aceleración es:

$$A_{out} = \frac{\frac{ADC_{value} \cdot 3{,}3}{1024} - 1{,}65}{0{,}330}$$

El ángulo de inclinación del dispositivo se puede inferir a partir de las mediciones de aceleración. Para los propósitos de esta demostración solo se computará el ángulo en el eje Y, Ψ (psi).

$$\Psi = atan(\frac{A_{yout}}{\sqrt{A_{xout}^2 + A_{zout}^2}})$$

El valor del ángulo calculado es un entero de 16 bits en el PIC. Para ser transferido a la GBA desde el PIC deberá partirse en dos transacciones puesto que el protocolo diseñado trabaja con mensajes de 8 bits. Esto se puede conseguir en C interpretando el valor del ángulo como un vector de bytes:

```
int* angulo;
unsigned char buffer[2];

[...]

acc2angle(angulo);

[...]

buffer[0] = *(((unsigned char*)angulo)+0);
buffer[1] = *(((unsigned char*)angulo)+1);

send_byte(buffer[0]);
send_byte(buffer[1]);
```

Cuando la GBA reciba los datos de las dos transacciones deberá hacer el proceso inverso.

Para mostrar el dato leído, la GBA pintará por pantalla un rectángulo que rotará según la inclinación del sistema periférico. El sistema enviará el ángulo leído una vez por segundo.

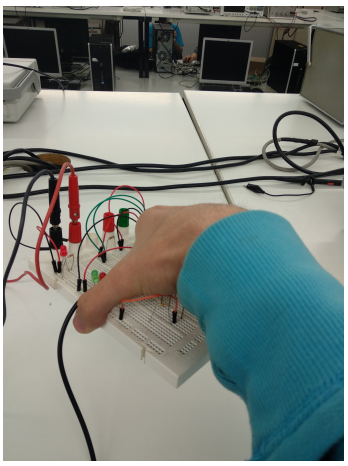

Figura 19: Sistema en reposo

Figura 20: Pantalla de la GBA con el rectángulo inclinado

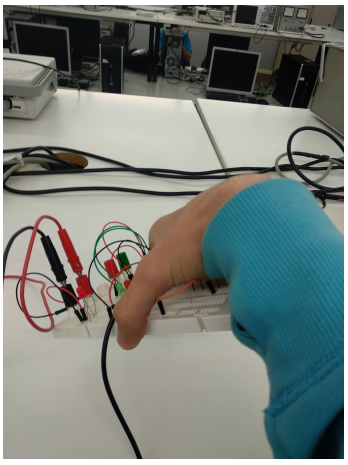

Figura 21: Sistema inclinado hacia adelante

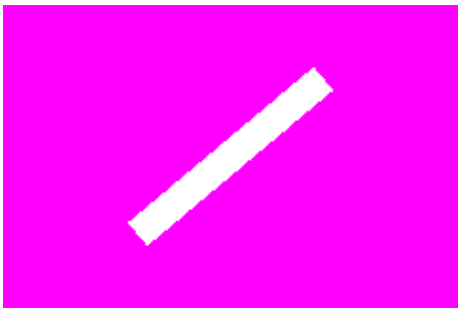

Figura 22: Pantalla de la GBA con el rectángulo inclinado

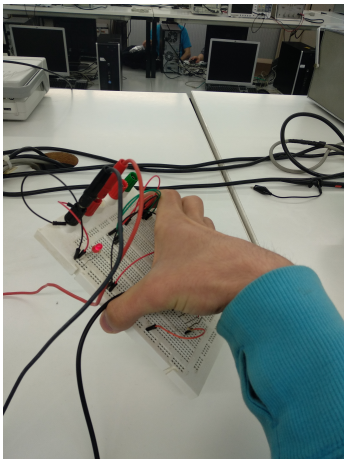

Figura 23: Sistema inclinado hacia atrás

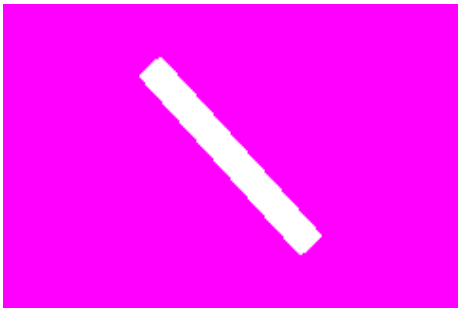

Figura 24: Pantalla de la GBA con el rectángulo inclinado

Gestión del proyecto

Definición de agentes

Jefe de proyecto

Encargado de la definición de objetivos y el cumplimiento de los mismos. Supervisa el resto de agentes y evalúa el progreso del proyecto.

Ingeniero hardware

Implementa el diseño funcional y los prototipos. Debe realizar sus tareas cumpliendo los requerimientos que el jefe de proyecto impone.

Tester

Se encarga de validar los prototipos creados por el ingeniero hardware. Su implicación a lo largo del proyecto ayuda al control de calidad del producto final.

Diseñador de producto

Presente en la etapa final del proyecto. Una vez definido el hardware del producto final, el diseñador debe idear un encapsulamiento que cumpla los requerimientos físicos del hardware y los aspectos de usabilidad que estime de cara al usuario final.

Definición de tareas

Definición de objetivos

El jefe de proyecto idea una lista de objetivos para el proyecto y define el alcance. Concreta entonces una planificación teniendo en cuenta las etapas del proyecto y los agentes implicados. Esta etapa dura 5 días y el riesgo de no cumplir el periodo estimado es muy bajo.

Propuesta de diseño

Una vez definidos los requerimientos del sistema a construir, el jefe de proyecto y en ingeniero hardware realizan una propuesta de diseño de manera conjunta. Es necesaria la colaboración de ambos agentes para asegurar que se cumplen los objetivos del proyecto así como se plantea un diseño coherente a nivel técnico. Esta etapa dura 5 días y el riesgo de no cumplir el periodo estimado es bajo.

Adquisición de material

Con la propuesta de diseño finalizada, el jefe de proyecto recaba el material necesario para dar comienzo al prototipado. Es su papel buscar y contactar con proveedores. Esta etapa es especialmente arriesgada porque el cumplimimento de las fechas depende de agentes externos al proyecto. Esta etapa dura 13 días y el riesgo de no cumplir con el periodo estimado es muy alto en cuanto

a la recepción del material y medio en cuanto a la evaluación del mismo. Para aliviar el riesgo de entregas con retraso se podrían programar de manera que se envíen escalonadamente. De esta manera se solapa la espera de un producto mientras se evalúa otro.

Prototipado

El ingeniero recibe todo el material necesario para comenzar el diseño físico y la implementación del prototipo. Al final de esta etapa se reúne con el tester para realizar pruebas básicas sobre el correcto funcionamiento del sistema. Esta etapa dura 14 días y el riesgo de no cumplir con el periodo estimado es medio. Para aliviar el riesgo se podrían incorporar al proyecto más ingenieros. De esta manera se divide el trabajo técnico entre más agentes pero aumenta el coste proporcionalmente.

Gestión de errores

El tester toma el prototipo creado por el ingeniero y comienza una etapa de prueba exhaustiva. Es de vital importancia que ambos agentes mantengan una comunicación permanente. Esta etapa dura 7 días y el riesgo de no cumplir con el periodo estimado es bajo.

Producción del producto final

Una vez probado el prototipo, el ingeniero hardware realiza un diseño de placa de producción que mandará a un agente externo para ser producida. Al mismo tiempo, el diseñador ideará y construirá el encapsulado del producto final. Esta etapa dura 10 días y el riesgo de no cumplir con el periodo estimado es bajo en cuanto a la parte que depende de los agentes internos pero medio en cuanto a la fase que implica agentes externos.

Programación temporal

En la Figura 25 se muestra el diagrama de Gantt del proyecto. Se ha dividido la ventana de tiempo en semanas. Cada tarea del proyecto tiene asociada un identificador de acuerdo con la Tabla 3 y en el diagrama se pinta con un color de fondo según su riesgo: verde, amarillo y rollo para representar poco, medio y gran riesgo respectivamente.

ID	Nombre	Duración
1	Definición de objetivos	5 días
2	Propuesta de diseño	5 días
3	Adquisición de material	13 días
3.1	Búsqueda y compra	2 días
3.2	Recepción	6 días
3.3	Comprobación del funcionamiento	5 días
4	Prototipado	14 días
4.1	Diseño funcional	4 días
4.2	Implementación física	5 días
4.3	Prueba de funcionamiento	5 días
5	Gestión de errores	7 días
6	Producción del producto final	10 días
6.1	Diseño físico de la placa	3 días
6.2	Diseño físico del encapsulado	3 días
6.3	Producción	7 días

Cuadro 3: Listado de tareas del proyecto

Figura 25: Diagrama de Gantt del proyecto

Metodología

Durante la programación de GEP se especificó que se usaría una metodología tipo SCRUM con Trello para hacer el seguimiento de las tareas del proyecto. A la hora de la verdad resultaba engorroso utilizar la herramienta online porque el laboratorio desde el que trabajaba no tenía una conexión estable en ese momento.

Mi metodología a la práctica era tener un archivo histórico que contiene todas las tareas realizadas durante la sesión de trabajo y por dónde retomarlo al comienzo de la siguiente sesión.

Gestión económica

Presupuesto de recursos

En esta sección se muestra una tabla por cada tipo de recurso con sus características más importantes como precio, vida útil, amortización y cantidad. Entonces se procederá a listar los costes indirectos relacionados al proyecto mediante otra tabla. Por último, se revisarán los imprevistos y contingencias para después calcular el presupuesto total.

Recursos humanos	**Dedicación (h)**	**Precio por hora (euro/h)**	**Coste total (euro)**
Jefe del proyecto	96	25	2400
Ingeniero hardware	216	20	4320
Tester	68	7.5	5100
Diseñador de producto	24	20	480
Total	**404**		**12300**

Cuadro 4: Costes de recursos humanos

En relación a los recursos humanos, las estimaciones de las horas dedicadas por cada agente implicado en el proyecto se han computado a partir de la planificación temporal expuesta en la Sección 8.3. Salvo el denominado tester, se asume que todos los agentes realizan una jornada laboral completa de 40 horas semanales.

Recurso software	**Precio (euro)**	Unidades
Ubuntu 17.04	0	1
Google Drive	0	1
EAGLE	0	1
Editores de texto:vim, sublime text, MPLAB	0	1
Compiladores: gcc, DevkitArm, xc compiler	0	1
Git	0	1
Firefox	0	1
Total	**0**	

Cuadro 5: Costes de recursos software

Costes indirectos

El proyecto se realizará en un apartamento personal. Por tanto, no hay costes añadidos por consumo energético o mobiliario.

Imprevistos y contingencias

Como se especificó en la Sección 8.2, las tareas con riesgo son:

Recurso hardware	Precio (euro)	Unidades	Vida útil
GameBoy Advance SP	49.99	1	4 años
Microcontrolador	1.75	1	4 años
Programador micro	167.81	1	4 años
Cable link	12.90	1	4 años
Sensores e integrados	50.00	1	4 años
Cableado	5.48	1	4 años
Placa de prototipado	3.31	1	4 años
Herramientas de soldado	19.98	1	4 años
Total	**311.22**		

Cuadro 6: Costes de recursos hardware

- 3.- Recepción de material y comprobación de su funcionamiento (riesgo medio y alto)
- 4.- Diseño y construcción del prototipo (riesgo medio)
- 5.- Producción del diseño final (riesgo medio)

En las tareas 3 y 5, el riesgo de incumplir el periodo estimado no comporta un impacto monetario porque se trata de un retraso debido a agentes externos al proyecto. El mayor problema que pueden causar es el retraso de otras tareas que dependan de ellas.

Por contra, el proceso de diseño y construcción del prototipo puede encontrarse con imprevistos durante el desarrollo que obligarían al ingeniero hardware a trabajar más horas de las previstas. Como medida contingente, se calcula que el ingeniero hardware trabajará un 15 % más, lo que supone: $0{,}15 \cdot 216h \cdot 20\ euro/h = 648$ euros de contingencia. Redondeando al alza, 650 euros.

Presupuesto final

A continuación se muestra una tabla englobando el presupuesto final del proyecto incluyendo las previsiones de cada recurso. Por brevedad, no aparecen en la tabla los recursos listados anteriormente que tengan coste cero.

Sostenibilidad y compromiso social

Matriz de sostenibilidad

	Proyecto en producción	Vida Útil	Riesgos
Ambiental	Consumo del diseño 7:10	Huella ecológica 10:20	Riesgos ambientales 0:0
Económico	Factura 5:10	Plan de viabilidad 10:20	Riesgos económicos 0:0
Social	Impacto personal 10:10	Impacto social 17:20	Riesgos sociales 0:0
Valoración total	22:30	37:60	0:0
			59:90

Cuadro 8: Matriz de sostenibilidad

Dimensión económica

Este proyecto nace de la dedicación de un pequeño equipo de entusiastas en el mundo de las consolas. Se ha estructurado de manera que simule la producción de un periférico como si una

Recursos humanos	
Jefe del proyecto	2400
Ingeniero hardware	4320
Tester	5100
Diseñador de producto	480
Total	**12300**
Recursos software	
Total	**0**
Recursos hardware	
GameBoy Advance SP	49.99
Microcontrolador	1.75
Programador micro	167.81
Cable link	12.90
Sensores e integrados	50.00
Cableado	5.48
Placa de prototipado	3.31
Herramientas de soldado	19.98
Total	**311.22**
Subtotal	**12611.22**
Total (21 % IVA)	**15259.58**

Cuadro 7: Costes Totales

pequeña empresa lo llevase a cabo. Por tanto, el cálculo de los costes hardware se ha estudiado en profundidad mientras que el cálculo de los recursos humanos ha sido más bien estimado.

El producto final no será comercializado por lo que no generará beneficio directo. No obstante, se podrá utilizar como material de soporte en actividades enfocadas al desarrollo de videojuegos para GameBoy Advance para aficionados.

Dimensión social

Como se mencionó en la Sección 1, uno de los objetivos del proyecto es contribuir a la comunidad aficionada a la GameBoy Advance con una documentación extensa y un nuevo producto que extiende las funcionalidades de la consola. Todos los integrantes de la comunidad se podrán ver beneficiados de este proyecto.

Dimensión ambiental

La selección de componentes y programación del firmware tiene en cuenta que la GameBoy Advance es una consola portátil y, como tal, tiene una autonomía limitada. Por tanto, debe considerarse el consumo energético del sistema. Esto tiene como consecuencia un menor impacto ambiental.

Conclusiones

Se han cumplido los objetivos?

Se fijó como objetivo comunicar la GBA con el microcontrolador del PIC a modo de puente con otros periféricos. La conexión se realiza mediante el cable propietario del fabricante de la consola. Una vez los dos dispositivos se pueden comunicar, la intención era extender las funcionalidades de la consola para incluir mecánicas nuevas de juego. En este sentido se ha cumplido el objetivo del proyecto. No obstante, solo se ha conectado la consola a un dispositivo periférico y no se ha hecho la parte del encapsulado final. El proyecto se ha quedado en la etapa de prototipado.

Se ha cumplido la planificación?

Las estapas de definición de objetivos y propuesta de diseño se cumplieron de acuerdo a la programación temporal.

Se realizó el inventario del material a adquirir y se encargó dentro del tiempo estimado. No obstante ocurrió una incidencia con uno de los elementos críticos: el cable serie oficial que utiliza la GBA tiene un conector con seis pines para utilizar todas las funcionalidades soportadas por la consola. Los cables disponibles en internet tienen conectores con cuatro pines y soportan un conjunto reducido de las funcionalidades. Esto supuso un retraso en la adquisición de material porque hubo que encontrar y encargar otro cable que tuviera los seis pines. El segundo cable resultó ser defecutoso por lo que hubo que volver al cable no oficial y desarrollar un protocolo alternativo de comunicación.

La etapa de producción del producto final no llegó nunca a ser debido a los contratiempos en las etapas anteriores. El proyecto debe terminar habiendo finalizado la etapa de prototipado y su respectivo control de errores.

Para acabar, la planificación temporal se hizo teniendo en cuenta que el proyecto estaba formado por varios agentes que trabajaban un determinado número de horas. A la hora de la verdad esto no es cierto: el proyecto lo he desarrollado solo. Desde el punto de vista de GEP la programación temporal está bien hecha pero no es realista. Esta es quizás la razón de mayor peso por la que no se ha cumplido con la planificación.

Próximos pasos

Puesto que el proyecto se ha quedado con un primer prototipo que utiliza un solo dispositivo yo recomiendo como próximo paso probar con mas dispositivos: módulo Wifi o GPS. Una vez se pruebe que el protocolo de comunicación es útil para distintas aplicaciones y dispositivos la etapa de prototipado quedará cerrada.

El último paso sería pulir uno de los prototipos para crear un producto final, tal y como se planificó durante la etapa de definición de objetivos.

Valoración personal

No he sido de seguir la planificación que realicé en GEP porque han surgido varios contratiempos que no supe prever. Aun así considero que el proyecto ha tomado un rumbo muy bueno. Es posible que siga con él en mi tiempo libre fuera del marco de la FIB.

Todo lo aprendido durante este proyecto podrá ser utilizado con certeza en futuros proyectos dentro del contexto de la asociación VGAFIB, que es donde se gestó la idea para este proyecto.

Referencias

[1] Videogames association fib. https://vgafib.com/. Accessed: 2018-04-12.

[2] Nintendo - gameboy advance. https://www.nintendo.com/consumer/systems/gameboyadvance/. Accessed: 2018-04-12.

[3] Curs de creació de videojocs per gameboy advance. https://vgafib.com/Cursos_d'estiu_2017. Accessed: 2018-04-12.

[4] Nintendo - gameboy advance and game boy advance sp accessories. https://www.nintendo.com/consumer/systems/gameboyadvance/accessories.jsp. Accessed: 2018-04-12.

[5] Gameboy advance wireless adapter. https://en.wikipedia.org/wiki/Game_Boy_Advance_Wireless_Adapter. Accessed: 2018-04-12.

[6] Yoshi's universal gravitation. https://en.wikipedia.org/wiki/Yoshi's_Universal_Gravitation. Accessed: 2018-04-12.

[7] gbadev.org. http://www.gbadev.org/. Accessed: 2018-04-12.

[8] Gameboy advance programming manual. https://www.scribd.com/document/198178973/GameBoy-Advance-Programming-Manual. Accessed: 2018-04-12.

[9] Programming the nintendo game boy advance: The unofficial guide. http://www.freeinfosociety.com/media/pdf/2901.pdf. Accessed: 2018-04-12.

[10] Gbatek. http://problemkaputt.de/gbatek.htm. Accessed: 2018-04-12.

[11] gpsvario. http://pataga.net/gpsvario.html. Accessed: 2018-04-12.

[12] Gameboy analog meter. https://sites.google.com/site/kenselectronicsprojects/gameboy-analog-meter. Accessed: 2018-04-12.

[13] Microchip - new/popular low power microcontrollers products. http://www.microchip.com/ParamChartSearch/chart.aspx?branchID=143. Accessed: 2018-04-16.

[14] Pic16f product brief. http://www.microchip.com/wwwproducts/en/PIC16F1709. Accessed: 2018-04-16.

[15] Pic24f product brief. http://www.microchip.com/wwwproducts/en/PIC24F08KL301. Accessed: 2018-04-16.

[16] Nintendo - gameboy. https://www.nintendo.com/consumer/systems/gameboy/index.jsp. Accessed: 2018-04-12.

[17] Gameboy advance. https://en.wikipedia.org/wiki/Game_Boy_Advance. Accessed: 2018-04-12.

[18] Nintendo - gameboy advance sp. https://www.nintendo.com/consumer/systems/gameboyadvance_sp/index.jsp. Accessed: 2018-04-12.

[19] Nintendo - gameboy micro. https://www.nintendo.com/consumer/systems/micro/index.jsp. Accessed: 2018-04-12.

[20] Nintendo - nintendo ds. https://www.nintendo.com/consumer/systems/ds/index.jsp. Accessed: 2018-04-12.

[21] Nintendo - connecting the nintendo gamecube gameboy advance link cable. https://www.nintendo.com/consumer/systems/nintendogamecube/gcn_agb_connect.jsp. Accessed: 2018-04-12.

[22] Tonc v1.4.2. http://www.coranac.com/tonc/text/toc.htm. Accessed: 2018-04-12.

[23] devkitpro. https://devkitpro.org/. Accessed: 2018-04-12.

[24] mgba. https://mgba.io/. Accessed: 2018-04-12.

[25] Ez-flash iv. http://www.ezflash.cn/product/ez-flash-iv/. Accessed: 2018-04-12.

[26] Accelerometer adxl335 interfacing with pic18f4550. http://www.electronicwings.com/pic/accelerometer-adxl335-interfacing-with-pic18f4550. Accessed: 2018-04-13.

[27] Adxl335 accelerometer module. http://www.electronicwings.com/sensors-modules/adxl335-accelerometer-module. Accessed: 2018-04-13.